操作系统实验指导

实验课题：同步机制及应用编程实现与比较

翟高寿

北京交通大学计算机学院

2021 年 2 月修订

1、实验目的

探索、理解并掌握操作系统同步机制的设计和实现机理，针对所谓的银行账户转账同步问题，构建基于 Peterson 算法的同步解决方案以及基于 Windows（或 Linux）操作系统同步机制（主要是互斥机制）的解决方案，并进行分析和比较。

2、实验内容

针对所谓的银行账户转账同步问题，分析、设计和利用 C 语言编程实现基于 Peterson 算法的同步解决方案，以及基于 Windows（或 Linux）操作系统同步机制的相应解决方案，并就自编同步机制与操作系统自身同步机制的效率进行比较和分析。

3、实验要求

同步机制及应用编程实现与比较实验功能设计要求：

（1）银行账户转账同步问题的抽象及未采取同步控制情况下的编程实现；

（2）基于 Peterson 算法的银行账户转账同步问题解决方案；

（3）基于 Windows（或 Linux）操作系统同步机制的银行账户转账同步问题解决方案；

（4）Peterson 算法同步机制和 Windows（或 Linux）操作系统同步机制的效率比较。

这里，银行账户转账假定为两个账户 nAccount1、nAccount2（均定义为初值是 0 的整型全局变量）之间进行，一个转出、一个转入，转账金额随机产生。整个程序共设立该两个账户之间的转账线程两个，且线程功能设计逻辑完全一致，不妨设转账线程的主体逻辑结构为如下的循环：

```
int nLoop = 0;
int nTemp1, nTemp2, nRandom;
do
{
    nRandom = rand();
    nTemp1 = nAccount1;
    nTemp2 = nAccount2;
    nAccount1 = nTemp1 + nRandom;
    nAccount2 = nTemp2 - nRandom;
```

```
    nLoop++;
} while ((nAccount1 + nAccount2) = = 0);
```

注意，在基于 Peterson 算法或 Windows（或 Linux）操作系统同步机制的银行账户转账同步问题解决方案中，上面的循环结束条件应修正为：

```
while(nLoop < 1000000);  或  while(nLoop < 5000000);
```

同时在该循环体中 nAccount2 = nTemp2 - nRandom;语句后插入如下条件语句：

```
if (nAccount1 + nAccount2 != 0)
  break;
```

实验报告撰写和提交要求：

（1）实验报告内容，须涵盖开发环境、运行环境、测试环境、源程序文件及源码清单、实验步骤、技术难点及解决方案、关键数据结构和算法流程、编译运行测试过程及结果截图、疑难解惑及经验教训、结论与体会等；

（2）在实验报告内容（如运行结果截图等适当位置）中应有机融入个人姓名、学号、计算机系统信息等凸显个人标记特征的信息；

（3）实验报告文档提交格式可为 Word 文档、WPS 文档或 PDF 文档。

4、成绩评价说明

本实验课题成绩评价满分按 5 分计。

实验课题得分根据自我独立完成情况、完成质量及实验报告水平综合决定。一般来说，获得满分要求有明确一致多项证据证实自我独立完成且满足实验课题所有要求。相反地，若无明确一致证据证实自我独立完成、甚至有明确证据证实存在抄袭行为，则酌情减分直至降为零分。

成绩评定细则指导建议如下：

（1）1 分：未采取同步控制情况下的银行账户转账同步问题的编程实现。

（2）1.5 分：基于 Peterson 算法的银行账户转账同步问题解决方案。

（3）1.5 分：基于 Windows（或 Linux）操作系统同步机制的银行账户转账同步问题解决方案。

（4）1 分：Peterson 算法同步机制和 Windows（或 Linux）操作系统同步机制的效率比较及结果分析。

（5）计算上述四项得分之和作为本实验课题成绩。

（6）互评成绩结果在提交慕课平台时按四舍五入取整处理。

5、实验编程提示-Windows 线程操作及同步机制相关函数

Windows 线程操作及同步机制所涉头文件包含语句、函数调用语句或函数原型如下：

（1）#include <windows.h>

（2）线程函数原型及框架

```
DWORD WINAPI ThreadExecutiveZGS(LPVOID lpParameter)
{
    int *pID = (int*)lpParameter;
    ……
    return 0;
}
```

（3）线程创建所涉相关变量及函数调用语句

```
HANDLE hThread[2];
int nPID0 = 0, nPID1 = 1;

if ((hThread[0] = CreateThread(NULL, 0, ThreadExecutiveZGS, &nPID0, 0, NULL)) == NULL)
{
    printf("线程 ThreadExecutiveZGS-0 创建失败！\n");
    exit(0);
}
if ((hThread[1] = CreateThread(NULL, 0, ThreadExecutiveZGS, &nPID1, 0, NULL)) == NULL)
{
    printf("线程 ThreadExecutiveZGS-1 创建失败！\n");
    exit(0);
}
```

（4）等待线程函数原型

```
DWORD WaitForMultipleObjects(
  DWORD nCount,
  CONST HANDLE *lpHandles,
  BOOL fWaitAll,
  DWORD dwMilliseconds
);
```

在主函数中调用本函数，以用来实现主线程对两个银行账户转账线程的等待：

```
WaitForMultipleObjects(2, hThread, TRUE, INFINITE);
```

（5）关于互斥信号量创建的函数原型及应用示例

```
HANDLE CreateMutex(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  BOOL bInitialOwner,
  LPCTSTR lpName
);

HANDLE hMutex = CreateMutex(NULL, FALSE, "MutexToProtectCriticalResource");
```

（6）关于互斥信号量申请（上锁/等待）的函数原型及应用示例

```
DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD dwMilliseconds
);

WaitForSingleObject(hMutex, INFINITE);
```

（7）关于互斥信号量释放（开锁/唤醒）的函数原型及应用示例

```
BOOL ReleaseMutex(
  HANDLE hMutex
);

ReleaseMutex(hMutex);
```

（8）关于线程挂起及线程句柄关闭操作的函数原型

```
VOID Sleep(DWORD dwMilliseconds);
BOOL CloseHandle(HANDLE hObject);
```

（9）关于系统时间获取的函数原型

```
DWORD GetTickCount(VOID); //返回值为从操作系统启动到当前时间所经过的毫秒数
```

6、实验编程提示-Linux 线程操作及同步机制相关函数

Linux 线程操作及同步机制所涉头文件包含语句、函数调用语句或函数原型如下：

（1）#include <pthread.h>

（2）编译命令示例（末尾须加上选项-lpthread）

gcc -o ThreadCSExclusion ThreadCriticalSectionExclusion.c **-lpthread**

（3）线程函数原型及框架示例

```
void* ThreadExecutiveZGS(void* zThreadName)
{
    char *pThreadName = (char*)zThreadName;
    ……
    return (void *)0;
}
```

（4）线程创建所涉函数及应用示例

```
int pthread_creat(pthread_t *tid, const pthread_attr_t *attr, void*(*start_routine)(void *), void *arg);
```

返回值：成功返回 0，错误返回错误号

第一个参数 tid：线程标识符（输出型参数）

第二个参数 attr：线程属性，一般设置为 NULL（表示线程属性取缺省值）

第三个参数 start_routine：函数指针，指向新线程即将执行的代码

第四个参数 arg：新线程对应函数的参数序列封装结果

```
pthread_t tid1;
if (pthread_create(&tid1, NULL, ThreadExecutiveZGS, "thread1"))
{
    printf("线程 ThreadExecutiveZGS-1 创建失败！\n");
    exit(0);
}
```

（5）关于等待线程的函数原型及应用示例

```
int pthread_join(pthread_t tid, void **rval_ptr);
```

返回值：成功返回 0，错误返回错误编号

第一个参数 tid：被等待线程的标识符（输入型参数）

第二个参数 rval_ptr：指向被等待线程对应函数的返回值

在主函数中调用本函数，以用来实现主线程对指定银行账户转账线程的等待：

```
void *ret1;
pthread_join(tid1, &ret1);
```

（6）关于互斥锁初始化、上锁/开锁的函数原型及应用示例

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/*上锁*/
if (pthread_mutex_lock(&mutex) != 0)
{
    printf("上锁失败！\n");
    exit(1);
}

/*开锁*/
if (pthread_mutex_unlock(&mutex) != 0)
{
    printf("开锁失败！\n ");
    exit(1);
}
```

（7）关于系统时间获取的函数原型

```
#include <time.h>
time_t time(time_t *t);
```

获取和返回从公元 1970 年 1 月 1 日的 0 时 0 分 0 秒【UTC 时间】算起到现在所经过的秒数。

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

获取系统当前精确时间（从公元 1970 年 1 月 1 日 0 时 0 分 0 秒 0 微秒【UTC 时间】算起到现在所经过的秒数和微秒数）存放于参数变量 tv 中（调用时相应时区信息参数变量一般置为 NULL）。其中时间信息结构体定义如下：

```
struct timeval
{
 long int     tv_sec;      /*秒数*/
 long int     tv_usec;     /*微秒数*/
};
```

5、国产平台鼓励说明

鼓励基于华为 OpenEuler 操作系统、龙芯 Loongson 操作系统等国产操作系统开展本实验课题的设计实现和测试验证，实验课题成绩及平时成绩评定将给予适当升档处理。对于北京交通大学的同学，可申请操作系统课程组华为泰山服务器（OpenEuler 操作系统）账号，亦可自主申请华为云虚拟机搭建 OpenEulerOS 等国产操作系统平台完成本实验课题。